Parallel Update Transactions
(Extended Abstract)
by
Dino Karabeg† Victor Vianu
Dept. of Computer Science and Engineering‡
Mail Code MC-014
University of California at San Diego
La Jolla, California 92093
U. S. A.

1. Introduction

While query languages have been extensively studied in the framework of the relational model, database
updates and transactions have only recently become the object of formal investigation. Indeed, most studies of
transactions have focused on concurrency issues [BG, PBR]. In [AV, KKPV], a formal model for sequential
update transactions in relational databases was introduced, and several basic results on transaction equivalence
and optimization were obtained. In the present paper we introduce a model for parallel update transactions,
which is an extension of the model developed in [AV] for sequential transactions. Our results focus on the prob-
lem of maximizing the degree of parallelism within parallel transactions, and producing optimal parallelizations
of sequential transactions.

Parallel transactions are viewed here as partially ordered sets of atomic database updates forming a seman-
tic unit. We consider a widely accepted class of atomic updates. These consist of insertions, deletions, and
modifications, where the selection of tuples (to be deleted or modified) involves the inspection of individual
attribute values for each tuple. We first look at the specification of parallel transactions. Since specifying a
parallel transaction by a partially ordered set of updates can be awkward, we propose a more convenient syntax,
called "in-line", and examine its relationship to partially ordered sets of updates. While the "in-line" syntax is
more restrictive, we show that it is powerful enough to capture the same relevant information on the degree of
parallelism as arbitrary partial orders of updates.

The main results of the paper focus on algorithms for maximizing the degree of parallelism within parallel
transactions. The algorithms can be used to optimize given parallel transactions, or to parallelize given sequen-
tial transactions. First, a notion of optimal parallel transaction is introduced. While it is shown that the optimi-
ization problem for parallel transactions is NP-complete, we exhibit efficient approximation algorithms which
produce parallelizations close to the optimum -- within a constant factor in general, and within absolute con-
stants (1 or 2) in special cases. The results for the special cases are of particular interest, since they provide new
examples of NP-complete problems with very good polynomial approximations. We argue that the same type
of problem is likely to occur in other contexts as well.

The paper consists of four sections. Section 2 summarizes the model for sequential transactions of [AV].
In Section 3, parallel transactions are defined and the optimization problem is shown to be NP-complete. In
Section 4 we present the approximate polynomial-time optimization algorithms.

2. Background on Sequential Transactions

In this section we review the model of sequential transactions and some results previously obtained in
[AV].

† On leave from Rudjer Boskovic Institute, Zagreb, Yugoslavia.
‡ The authors were supported in part by the National Science Foundation, under grant number IST-8511538.
We assume knowledge of the basic concepts and notation of relational databases, as in [M, U].

The sequential transactions we consider are finite sequences of insertions, deletions, and modifications. We focus on the large class of "domain-based" transactions, where the selection of tuples to be deleted or modified involves the inspection of individual attribute values of a tuple, independently of other attribute values in the tuple and of other tuples in the relation.

The following is a simple example of a domain-based transaction in SQL [D].

2.1 Example. Suppose a relation EMP (employee) has been defined (its attributes are NAME, DEPT, RANK, and SALARY). The following transaction hires Moe as the new manager of the parts department, with a salary of 30K, then fires all managers from the parts department other than Moe. Finally, all employees from the parts department who are not managers are transferred to the service department. The rank remains unchanged. The new salary is 20K:

- insert into EMP values ('moe','parts','manager',30K)
- delete from EMP where NAME # 'moe' and DEPT = 'parts' and RANK = 'manager'
- update EMP set DEPT = 'service', SALARY = 20K where DEPT = 'parts' and RANK # 'manager'. □

We now define the notions of a "condition" and satisfaction of a condition by a tuple.

Definition. Let U be a set of attributes. A condition over U is an expression of the form A=a or A# a, where A ∈ U and a ∈ dom(A). A tuple u over U satisfies a condition A=a (A# a) iff u(A) = a (u(A) ≠ a). A tuple u satisfies a set C of conditions if it satisfies every condition in C. We do not explicitly use logical connectors to build up complex conditions. It can be easily seen that this would not add power to our transactions. In the following, only satisfiable sets of conditions are considered, that is, sets of conditions with no mutually exclusive conditions. Although the conditions use only equality and inequality, this assumption is not central to the development (it is straightforward to extend the conditions so that comparisons of the form A>a, A<a are allowed).

A set of conditions over U is used to specify a set of tuples over U (those satisfying the conditions). Due to the form of our conditions, we use the intuitively suggestive term "hyperplane" to identify such sets of tuples:

Definition The hyperplane H(U,C) defined by a (satisfiable) set C of conditions over U is the set \( \{ t ∈ Tup(U) \mid t \text{ satisfies } C \} \).

For simplicity, we sometimes use the same notation for a set C of conditions over U and for the hyperplane H(U,C) defined by C. Thus, we say "hyperplane C" instead of "hyperplane H(U,C)". Whenever U is understood. The support of a hyperplane H(U,C) is the set of attributes \( \{ A \mid A = a \text{ is in } C \text{ for some } a \} \). Thus the support of the hyperplane defined by \( \{ A=0, B=1, C=5 \} \) is AB.

We now define the updates used to build our transactions. An insertion over a database schema R is an expression \( i_X(C) \) where X is a relation schema in R and C is a set of conditions specifying a complete tuple over X. A deletion over R is an expression \( d_X(C) \), where X is a relation schema in R and C is a set of conditions over X. Finally, a modification over R is an expression \( m_X(C_1; C_2) \), where X is a relation schema in R, C_1 and C_2 are sets of conditions over X and, for each A in X, either (1) \( C_1|_A = C_2|_A \) or \( A = a ∈ C_2 \). (The equalities present in C_2 but not in C_1 indicate how tuples in \( H(X, C_1) \) are modified.) Note that, if m(C_1; C_2)